Ada Issue 00422 User Data for Timing Events

!standard D.15(1) 05-03-09 AI95-00422/01
!class amendment 05-03-09
!status work item 05-03-09
!status received 05-03-09
!priority High
!difficulty Easy
!subject User data for timing events

!summary

!problem

The specification of package Timing_Events declares a limited private type Timing_Event, together with a handler type:

```
package Ada.Real_Time.Timing_Events is
  type Timing_Event is limited private;
  type Timing_Event_Handler
       is access protected procedure (Event : in out Timing_Event);
  ...
end Ada.Real_Time.Timing_Events;
```

When the handler is called, it is passed the event that fired. Unfortunately, there is very little that the handler can do with this event parameter, other than calling some operations from this package.

It would seem that a reasonable usage paradigm would be for a single handler to handle several events. Here is an example:

```
Eggs_Done : array (1 .. 10) of Timing_Event;

protected Egg is
  procedure Is_Done (Event: in out Timing_Event);
end Egg;

protected body Egg is
  procedure Is_Done (Event: in out Timing_Event) is
  begin
    -- How do we know which egg is done here?
  end Is_Done;
end Egg;
```

and then in various tasks do:

```
Put_Egg_In_Water;
Set_Handler (Event => Egg_Done (I),
       In_Time => ...,
            Handler => Egg.Is_Done'Access);
```

If we have ten eggs being cooked asynchronously, how do we know which one is done when we enter the handler? One could naively try:

```
for I in Eggs_Done'Range loop
   if Eggs_Done (I) = Event then
      -- This is the one.
   end if;
end loop;
```

but this doesn't work because Event is limited. Changing the test to:

```
   if Eggs_Done (I)'Access = Event'Access then
```

doesn't work because Event is not aliased (the components of Eggs_Done could be made aliased). Using Address would be legal, but it would cause a horrendous dependency on the parameter passing mode for events.

So in order to determine which event fired, it seems necessary to have as many protected procedures as events. That seems unwieldy, and possibly expensive in terms of resources.

!proposal

One could consider making Timing_Event nonlimited, but it seems undesirable for a type whose objects are going to be owned by the runtime system. Having user copy them around would probably cause an undue implementation burden.

We propose to use an access discriminant to associate some user-specific information with a timing event. A new predefined unit, Generic_Timing_Events, is added, whose specification looks like:

```
generic
   type User_Data (<>) is limited private;
package Ada.Real_Time.Generic_Timing_Events is
  type Timing_Event (Data : access User_Data := null) is limited private;
  type Timing_Event_Handler
       is access protected procedure (Event : in out Timing_Event);
  ... -- As before
end Ada.Real_Time.Generic_Timing_Events;
```

The description of the semantics of Timing_Events becomes the description of this new package, and Timing_Events is made a nongeneric equivalent of Generic_Timing_Events with actual type Integer. (Why Integer? Because we need some actual type for User_Data, and Integer seems rather innocuous. An alternative would be to declare a null record somewhere and use it in the nongeneric equivalent, but that's probably overkill.)

!wording

1 - D.15 must show the specification of Generic_Timing_Events, and must contain a blurb explaining the purpose of the formal type User_Data.

2 - A sentence must be added after the (generic) package specification explaining the existing of the nongeneric equivalent.]

!discussion

!example

With this proposal, we'd create an appropriate Timing_Event instance:

```
  package Egg_Event is new Generic_Timing_Event (Natural);
```

and then we could write the handler as:

```
  protected body Egg is
     procedure Is_Done (Event: in out Egg_Event.Timing_Event) is
     begin
        -- Event.Data indicates the correct egg.
     end Is_Done;
  end Egg;
```

Each use would look like:

```
  Egg1 : constant aliased Natural := 1;
  Egg_Done1 : Egg_Event.Timing_Event(Egg1'access);

  ...
  Put_Egg_In_Water;
  Set_Handler (Event => Egg_Done1,
               In_Time => ...,
                    Handler => Egg.Is_Done'Access);
```

--! corrigendum

!ACATS test

!appendix